INDEXING METHOD AND APPARATUS

The present invention relates to an apparatus and method for indexing sequences of sub-word units, such as sequences of phonemes or the like. The invention can be used to identify regions of a database for search in response to a user's input query. The input query may be a voiced or typed query.

Databases of information are well known and suffer from the problem of how to locate and retrieve the desired information from the database quickly and efficiently. Existing database search tools allow the user to search the database using typed key words. Whilst this is quick and efficient, this type of searching is not suitable for various kinds of databases, such as video or audio databases.

A recent proposal has been made to annotate such video and audio databases with a phonetic transcription of the speech content of the audio and video files, with subsequent retrieval being achieved by comparing a phonetic transcription of the user's input query with the phoneme annotation data in the database. The technique proposed for matching the sequences of phonemes firstly

defines a set of features in the query, each feature being taken as an overlapping fixed size fragment from the phoneme string. It then identifies the frequency of occurrence of the features in both the query and the annotation and then finally determines a measure of the similarity between the query and the annotation using a cosine measure of these frequencies of occurrences.

However, as those skilled in the art will appreciate, if the database is large, then this retrieval method becomes unfeasibly long. An indexing method is therefore required.

As is well known, indexing provides a one-to-many mapping between the index (sometimes referred to as the key) and the data in the database. Where the database comprises words, this indexing is simple, but for phonemes this raises a number of difficulties. Firstly, because there are only a small number of phonemes (approximately 43 in the English language) means that a naive mapping using a single phoneme as a key is not sufficiently discriminating, since any given phoneme will occur several thousand times in the database. Secondly, because of the relatively poor recognition rate of phonemes (60% to 70%) means that any one-to-many mapping

will make it difficult to retrieve data where the query phoneme or annotation phoneme was misrecognised, inserted or omitted. Finally, performing any statistical retrieval methods becomes computationally unfeasible.

The present invention aims to provide an efficient sub-word indexing technique which can be used in a retrieval system to identify areas of a database for searching.

Exemplary embodiments of the present invention will now be described with reference to the accompanying drawings, in which:

Figure 1 is a schematic block diagram illustrating a user terminal which allows the user to retrieve information from an input typed or voice query;

Figure 2 is a schematic diagram of phoneme and word lattice annotation data which is generated from a voiced input by the user for annotating a document;

Figure 3a diagrammatically illustrates the block nature of an annotation stored in the annotation database which forms part of the user terminal shown in Figure 1;

Figure 3b is a schematic diagram illustrating a sequence of annotation phonemes which is included in one of the blocks of the annotation shown in Figure 3a;

Figure 3c schematically illustrates a sequence of phoneme clusters for the phoneme sequence shown in Figure 3b and illustrates how these phoneme clusters can be grouped to form a number of overlapping phoneme cluster N-grams;

Figure 4 is a flowchart illustrating the main processing steps involved in creating a phoneme index;

Figure 5 illustrates an example of a phoneme index which is generated during the processing of the steps shown in Figure 4;

Figure 6 is a flowchart illustrating the processing steps involved in performing a phoneme search of an annotation database;

Figure 7a schematically illustrates a sequence of phonemes representing an input query;

Figure 7b schematically illustrates the way in which the sequence of phonemes shown in Figure 7a can be divided

into a number of overlapping phoneme N-grams;

Figure 7c illustrates a number of overlapping phoneme cluster N-grams derived from the phoneme N-grams shown in Figure 7b;

Figure 8 is a flowchart illustrating the main processing steps involved in using the phoneme index to identify locations of the annotation for phoneme matching;

Figure 9a is a flowchart illustrating part of the process steps involved in determining the different phoneme clusters;

Figure 9b is a flowchart illustrating the remaining process steps involved in determining the different phoneme clusters;

Figure 10 is a schematic block diagram illustrating the form of an alternative user terminal which is operable to retrieve a data file from a database located within a remote server in response to an input voice query; and

Figure 11 illustrates another user terminal which allows a user to retrieve data from a database located within a

remote server in response to an input voice query.

Embodiments of the present invention can be implemented using dedicated hardware circuits, but the embodiment to be described is implemented in computer software or code, which is run in conjunction with processing hardware such as a personal computer, work station, photocopier, facsimile machine, personal digital assistant (PDA), web browser or the like.

DATA FILE RETRIEVAL

Figure 1 is a block diagram illustrating the form of a user terminal 59 which is used, in this embodiment, to retrieve documents from a document database 29 in response to a voice or typed query input by the user 39. The "document" may be text documents, audio files, video files, photographs, mixtures of these etc. The user terminal 59 may be, for example, a personal computer, a hand-held device or the like. As shown, the user terminal 59 comprises the document database 29, an annotation database 31 comprising a descriptive annotation of each of the documents in the document database 29, a phoneme matcher 33, a phoneme index 35, a word matcher 37, a word index 38, a combiner unit 40, an automatic speech recognition unit 51, a phonetic

transcription unit 75, a keyboard 3, a microphone 7 and a display 57.

In operation, the user inputs either a voice query via the microphone 7 or a typed query via the keyboard 3 and the query is processed either by the automatic speech recognition unit 51 or the phonetic transcription unit 75 to generate corresponding phoneme and word data. The phoneme data is input to the phoneme matcher 33 which is operable to perform a phoneme search in the annotation database 31 with reference to a phoneme index 35. Similarly, the word data is input to the word matcher 37 which is operable to search the annotation database 31 with reference to the word index 38. The results of the phoneme and word search of the annotation database are then input to the combiner unit 40 which uses these results to retrieve a ranked list of documents 41 from the document database 29 which are output to the display unit 57 for display to the user 39.

In this embodiment, the annotation data for each document comprises a combined phoneme (or phoneme-like) and word lattice. Figure 2 illustrates the form of the phoneme and word lattice annotation data generated for the spoken annotation "picture of the Taj Mahal". As shown, the

phoneme and word lattice is an acyclic directed graph with a single entry point and a single exit point. It represents different parses of the user's input. It is not simply a sequence of words with alternatives, since each word does not have to be replaced by a single alternative, one word can be substituted for two or more words or phonemes, and the whole structure can form a substitution for one or more words or phonemes. Therefore, the density of the data within the phoneme and word lattice annotation data essentially remains linear throughout the annotation data, rather than growing exponentially as in the case of a system which generates the N-best word lists for the annotation input.

In this embodiment, the annotation data for each document (d) is stored in the annotation database 31 and has the following general form:

HEADER

- flag if word if phoneme if mixed
- time index associating the location of blocks of annotation data within memory to a given time point.
- word set used (i.e. the dictionary)
- phoneme set used

- the language to which the vocabulary pertains
- phoneme probability data

Block(i) i = 0,1,2,.....

node n_j j = 0,1,2,.....

- time offset of node from start of block
- phoneme links (k) k = 0,1,2.....
  offset to node n_j = n_k-n_j (n_k is node to which link K extends) or if n_k is in block(i+1) offset to node n_j = n_k+N_b-n_j (where N_b is the number of nodes in block(i))
  phoneme associated with link (k)
- word links (l) l = 0,1,2,.....
  offset to node n_j = n_i - n_j (n_j is node to which link l extends) or if n_k is in block(i+1) offset to node n_j = n_k+N_b-n_j (where N_b is the number of nodes in block(i))
  word associated with link (l)

The flag identifying if the annotation data is word annotation data, phoneme annotation data or if it is mixed is provided since the annotation data may include just word data, just phoneme data or both word and

phoneme data.

In this embodiment the annotation data is divided into blocks (B) of nodes (n) in order to allow the search to jump into the middle of the annotation data. The header therefore includes a time index which associates the location of the blocks of annotation data within the memory to a given time offset between the time of start and the time corresponding to the beginning of the block.

The header also includes data defining the word set used (i.e. the dictionary), the phoneme set used and their probabilities and the language to which the vocabulary pertains. The header may also include details of the automatic speech recognition system or the phonetic transcription system used to generate the annotation data and any appropriate settings thereof which were used during the generation of the annotation data.

The blocks of annotation data then follow the header and identify, for each node in the block, the time offset of the node from the start of the block, the phoneme links which connect that node to other nodes by phonemes and word links which connect that node to other nodes by words. Each phoneme link and word link identifies the

phoneme or word which is associated with the link. They also identify the offset to the current node. For example, if node n_{50} is linked to node n_{55} by a phoneme link, then the offset to node n_{50} is 5. As those skilled in the art will appreciate, using an offset indication like this allows the division of the continuous annotation data into separate blocks.

In an embodiment where an automatic speech recognition unit outputs weightings indicative of the confidence of the speech recognition unit's output, these weightings or confidence scores would also be included within the data structure. In particular, a confidence score would be provided for each node which is indicative of the confidence of arriving at the node and each of the phoneme and word links would include a transition score depending upon the weighting given to the corresponding phoneme or word. These weightings would then be used to control the search and retrieval of the data files by discarding those matches which have a low confidence score.

In order to provide an efficient retrieval method, a word indexing scheme and a phoneme indexing scheme is used in order to identify portions in the annotation database 31

against which a direct comparison with the input query is made. The word index 38 and the way that it is used is well known to those skilled in the art and will not be described further. However, the way in which the phoneme index 35 is generated and subsequently used to identify portions of the annotation database 31 for comparison with the input query will now be described in more detail.

As mentioned above, the use of a single phoneme as the key for a phoneme index will not provide sufficient discrimination, since each phoneme will occur several thousand times in the annotation database 31. Further, since current automatic speech recognition systems have a relatively poor phoneme recognition rate (60 to 70%), indexing using the phonemes directly will make it difficult to retrieve the data where the query phoneme or the annotation phoneme was misrecognised. Since the automatic speech recognition system tends to produce decoding errors for similar sounding phonemes, such as /s/ and /z/ and not highly dissimilar phonemes, such as /z/ and /g/, the error rate of indexing can be greatly reduced by indexing on confusable clusters of phonemes rather than individual phonemes.

Considering, for example, the situation where a query and an annotation exist for the word "sheep" and the query (Q) comprises the sequence of phonemes /sh//iy//p/ and the annotation (A) comprises the sequence of phonemes /s//eh//p/. If the sequence of three query phonemes is used as an index into the annotation (in the form of a trigram), then it will not be possible to retrieve the data since the sequence of query phonemes does not match the sequence of annotation phonemes. However, if the phonemes are clustered into confusable sets, and the phonemes in the query and in the annotation are classified into their respective sets, then there is a better chance that there will be a match between the query and the annotation if they both sound alike. For example, if the following phoneme classifications are defined:

C_1 = {/s/ /z/ /sh/ /zh/}

C_2 = {/t/ /k/ /g/ /b/ /p/}

⋮

C_5 = {/eh/ /ih/ /iy/}

and the query phonemes and the annotation phonemes in the above illustration are classified into these classes, then this will result in the following cluster trigrams:

$C(Q) = \{C_1\ C_5\ C_2\}$

$C(A) = \{C_1\ C_5\ C_2\}$

Therefore, matching using the clustered query and annotation will now work since $C(Q) = C(A)$ and the data can be retrieved.

In this embodiment, a hash indexing technique is used which tries to create a unique mapping between a key and an entry in a list. In this embodiment trigrams of the above phoneme clusters are used as the key to the index. The way that the phoneme index 35 is created and used will now be described in more detail.

In order to create the phoneme index 35 the phoneme annotation data stored in the annotation database 31 is converted into phoneme cluster trigrams. The way that this is achieved is illustrated in Figure 3. In particular, Figure 3a schematically illustrates the block form of the annotation data which is stored in the database 31 for one of the documents (d) stored in the document database 29. As shown, the annotation data comprises successive blocks of data B_0^d to B_{M-1}^d. As mentioned above, the annotation data within each block includes the nodes within the block and the phoneme and word links which are associated with the nodes. In order to illustrate the indexing method used in this embodiment, the remaining description will assume that

the annotation data for document d includes a canonical sequence of phonemes i.e. one with no alternatives. Figure 3b illustrates the canonical sequence of phonemes within the i^{th} block for the annotation data for document d. As shown, block B^d_i comprises the canonical sequence of phonemes a_0^{di} to a_{Ndi}^{di} which extend between nodes n_0^{di} to n_{Ndi}^{di}. Figure 3c illustrates the overlapping "cluster trigrams" 101 generated for the sequence of annotation phonemes in block B^d_i shown in Figure 3b. As shown in Figure 3c, the cluster in which each of the annotation phonemes in block B^d_i belongs is determined. Then cluster trigrams are determined from overlapping groups of three cluster identifications for the sequence of annotation phonemes. In particular, the first cluster trigram determined is $C(a^{di}_0)\ C(a^{di}_1)\ C(a^{di}_2)$ then cluster trigram $C(a^{di}_1)\ C(a^{di}_2)\ C(a^{di}_3)$ etc. Although not shown in Figure 3c, it is also necessary to consider the trigrams which bridge adjacent blocks. For example, in this embodiment, it would be necessary to consider the following cluster trigrams: $C(a^{di-1}_{N(di-1)-1})\ C(a^{di-1}_{Ndi-1})\ C(a^{di}_0)$ and cluster trigram $C(a^{di-1}_{Ndi-1})\ C(a^{di}_0)\ C(a^{di}_1)$.

To create the index, a large table or array, A, having S entries is created. In this embodiment, each entry is addressed by an index (IDX) which takes a value between

zero and S-1 and each entry includes a data field to store the key (KEY) associated with the entry and a data field to store the pointers which identify the relevant locations within the annotation database where the phoneme data associated with the KEY can be found. Initially each of the entries is empty. The size of the table depends on the number of different phoneme clusters and the number of cluster identifications in the key. In this case, three cluster identifications (trigrams) are provided in each key. If there are ten clusters, then the number of different possible keys is 3^{10}. Therefore, in this case, S should, theoretically be made approximately equal to 3^{10}. However, in practice, some of the possible keys are unlikely to occur. Therefore, the size of the index can be set to have some initial size and data can be added to the index until more than a predetermined percentage of the entries are full. At this point, a bigger table can be created and the data from the old table copied over to the new table. This process can be repeated until there is no more data to be added to the index. Although this means that some memory is wasted, this is insignificant compared to the memory required for the pointers which will be stored in the table.

The way that data is added to the table will now be explained with reference to Figure 4. As shown, in step s1, the system calculates the value of a function (f(KEY)) which is dependent upon the key, i.e. a function of the current cluster trigram, which value is used as the index (IDX) into the table A. In particular, the function f(KEY) defines a mapping between the cluster trigram and an entry in the table A and always yields a number between zero and S-1. In this embodiment, the function used is:

$$[C[1]K_c\ C[2]K_c\ C[3]K_c] \bmod S \qquad (1)$$

where K_c is the number of phoneme clusters and C[1] is the number of the cluster to which the first annotation phoneme in the trigram belongs, C[2] is the number of the cluster to which the second annotation phoneme in the trigram belongs and C[3] is the number of the cluster to which the third annotation phoneme belongs. For example, for the illustration above where $C(A) = \{c_1\ c_5\ c_2\}$, C[1] = 1, C[2] = 5 and C[3] = 2.

Once IDX has been determined in step s1, the processing proceeds to step s3 where the system checks the corresponding entry in the table, A, and determines

whether or not the key stored in that entry (KEY) is the same as the key for the current phoneme cluster trigram (key) or is the null key (indicating that this entry is empty). If in step s3 the system determines that the key stored in the entry (KEY) matches the current key (key) or the null key, then the processing proceeds to step s5 where a pointer is added to that entry of the table, A, which points to the node associated with the first phoneme in the phoneme trigram associated with the current input key. For example, for the key $c(a_0^{di})$ $c(a_1^{di})$ $c(a_2^{di})$, the data which would be added to the table would be a pointer which points to the node n_0^{di} since annotation phoneme a_0^{di} is associated with node n_0^{di}. If the key stored in the entry is currently the null key, then in step s5, the system also changes the key for the entry (KEY) to the current key (key). The processing of the current cluster trigram then ends and a similar processing is performed on the next cluster trigram. If at step s3, the processing determines that the IDX^{th} entry in the table, A, has already been assigned to a different cluster trigram, then the processing proceeds to step s7 where the system tries another entry in the table by changing the value of IDX in some predetermined way. In this embodiment, this is achieved by calculating:

$$IDX = (IDX + V) \bmod S \qquad (2)$$

where V is some fixed number which is not a factor of S (other than 1). The reason that V should not be a factor of S is that this ensures that all entries in the table are tried. For example, if S = 10 and V = 2 then this technique would simply keep trying either just the odd or just the even entries of table A. In order to avoid this problem, S should preferably be prime. After step s7, the processing returns to step s3.

Once all the cluster trigrams in all the blocks of each annotation have been processed in this way, the table, A, is stored as the phoneme index 35. Figure 5 illustrates the form of a phoneme index that is generated by the above processing. As can be seen from Figure 5, each entry in the table includes the index number (IDX) of the entry, the key (KEY) associated with the entry and (if the key is not the null key) one or more pointers pointing to nodes in the annotation database 31. As shown, in this embodiment, these pointers have the form n[p,q,r] where p is the annotation for document p, q is the q^{th} block of nodes within that annotation data and r is the r^{th} node within that block.

The way that the phoneme matcher 33 uses the phoneme index 35 in response to an input query in order to identify portions of the annotation database 31 for matching with the input query will now be described with reference to Figures 6 to 8.

When a user inputs a query the phoneme data generated either by the automatic speech recognition unit 51 or the phonetic transcription unit 75 (depending upon whether the input query was received through the microphone or through the keyboard) is input to the phoneme matcher 33. Figure 6 illustrates the processing steps performed by the phoneme matcher 33 on this phoneme data. As shown, in step s11, the phoneme matcher 33 converts the received query phoneme data into overlapping phoneme trigrams. Figure 7a illustrates a sequence of query phonemes q_0 to q_5 representative of phoneme data received by the phoneme matcher 33 and Figure 7b illustrates how this sequence of query phonemes is converted into five overlapping trigrams of query phonemes 103. The processing then proceeds to step s13 where each of the query phoneme trigrams is converted into phoneme cluster trigrams 105, as illustrated in Figure 7c, by classifying each of the query phonemes in the phoneme trigram into one of the above classes or clusters. The processing then proceeds

to step s15 where the phoneme matcher 33 uses each of the cluster trigrams generated for the input query to address the phoneme index 35 in order to identify relevant locations in the annotation database 31.

Figure 8 illustrates the processing steps used by the phoneme matcher 33 in carrying out step s15. As shown, in step s151, the phoneme matcher 33 calculates the index (IDX) for the entry in the table, A, by inserting the current query cluster trigram into the function defined above in equation (1). The processing then proceeds to step s153 where the phoneme matcher 33 checks the corresponding entry in the table, A, and determines whether or not the key stored in that entry (KEY) is the null key (indicating that this entry is empty). If it is, then the processing proceeds to step s155 where the phoneme matcher 33 determines that there is no corresponding annotation in the annotation database and outputs an appropriate output to the combiner unit 40. The processing then ends.

If at step s153 the phoneme matcher 33 determines that the key stored in the entry (KEY) is not equal to the null key, the processing proceeds to step s157 where the phoneme matcher 33 determines whether or not the key

stored in the entry (KEY) is the same as the key for the current query cluster trigram (key). If it is then the processing proceeds to step s159 where the phoneme matcher 33 retrieves the pointers from that entry. The processing then ends. If, however, the phoneme matcher 33 determines, in step s157, that the key for the entry (KEY) does not equal the key for the current query cluster trigram (key), then the processing proceeds to step s161 where the phoneme matcher tries another entry in the table by changing the value of the index (IDX) using equation (2) given above and then returning to step s153.

Once the phoneme matcher 33 retrieves the pointers from the index or determines that there is no data stored for the current query cluster trigram, the phoneme matcher 33 then performs a similar processing for the next query cluster trigram until all the query cluster trigrams have been processed in this way. The processing then proceeds to step s17 shown in Figure 6, where the phoneme matcher 33 uses the pointers identified in step s15 to identify regions within the annotation database 31 which will be matched with the actual phoneme data received by the phoneme matcher 33. In this embodiment, these regions are identified by comparing the pointers retrieved in

step s159 for successive query cluster trigrams and looking for pointers which point to portions in the annotation database 31 which are next to each other. For example, referring to the phoneme index illustrated in Figure 5, if the n^{th} query cluster trigram is $c_5c_3c_6$ and the $n+1^{th}$ query cluster trigram is $c_3c_6c_1$, then the phoneme matcher 33 will identify node 40 of the 32^{nd} block of annotation data for the 3^{rd} document as being a region of the annotation database for further processing in step s19. This is because the pointers stored in the phoneme index 35 for the key $c_5c_3c_6$ includes a pointer to node n[3,32,40] and the pointers stored in the phoneme index 35 for the key $c_3c_6c_1$ includes a pointer to node n[3,32,41], which is immediately after node n[3,32,40] and is therefore consistent with a portion of an annotation having successive cluster trigrams $c_5c_3c_6$ and then $c_3c_6c_1$ which may match with the input query.

After the phoneme matcher 33 has identified the regions in step s17, it performs a phoneme comparison between the received query phoneme data and the phoneme data stored in the annotation database 31 at the regions identified in step s17. This phoneme comparison can be performed by comparing M-grams of the query with similar M-grams of the annotation (as described in the applicant's earlier

UK application GB 9905201.1, the content of which is incorporated herein by reference) or by performing a dynamic programming comparison between the sequence of query phonemes and the sequence of annotation phonemes (using, for example, one of the techniques described in the applicant's earlier UK application GB 9925574.7, the content of which is incorporated herein by reference). The results of these phoneme comparisons are then output to the combiner unit 40 where the results are combined with the output from the word matcher 37 in order to retrieve and rank the appropriate documents from the document database 29.

In the above description, it has been assumed that the phonemes have been classified into a number of sets of confusable phonemes. The way that these phoneme clusters are determined in this embodiment will now be described. If two phoneme decodings have been made, once during the annotation phase and once during the query phase, then the probability of the two decodings, p_1 and p_2, coming from the same source is given by:

$$P(p_1,p_2|m_1,m_2) = \sum_x P(p_1|x,m_1)P(p_2|x,m_2)P(x) \qquad (3)$$

where P(p|x,m) is the probability of decoding phoneme x

as phoneme p when decoding method m is used and P(x) is the probability of phoneme x occurring. The decoding methods m_1 and m_2 need to be distinguished since one of the decodings may come from a text-to-phoneme converter within the phonetic transcription unit 75 whilst the other may come from the automatic speech recognition unit 51 and these two different decoding techniques will suffer from different types of confusion. Each of these probabilities in equation (3) above can be determined in advance during a training routine by applying known speech to the automatic speech recognition unit 51 and known text into the phonetic transcription unit 75 and by monitoring the output from the recognition unit 51 and the phonetic transcription unit 75 respectively. The way that such a training routine would be performed will be well known to those skilled in the art and will not, therefore, be described further here.

If it is assumed that there are K_c phoneme clusters or classes and for each cluster there is a many-to-one mapping between decoded phonemes and the clusters. This mapping will depend on the decoding method employed. For example, an /s/ decoded by the automatic speech recognition unit 51 may be in cluster c_4, but an /s/ decoded by the phonetic transcription unit 75 may be in

cluster c_2. Therefore, for any particular cluster (K_i) there will be a probability that two decodings from the same source are classed into that cluster which is determined by summing the probability given in equation (3) above for all possible combinations of decodings, p_1 and p_2, which belong to that cluster, i.e. by calculating:

$$P\left(\begin{matrix}\text{Assigned to}\\ \text{same cluster}\end{matrix}\middle| K_i\right) = \sum_{p_1 \in K_i} \sum_{p_2 \in K_i} P(p_1,p_2|m_1,m_2) \qquad (4)$$

The probability that all decodings are correctly classified is therefore given by:

$$P\left(\begin{matrix}\text{All correctly}\\ \text{classified}\end{matrix}\right) = \prod_{i=1}^{K_c} P\left(\begin{matrix}\text{Assigned to}\\ \text{same cluster}\end{matrix}\middle| K_i\right) \qquad (5)$$

The task of defining the clusters aims, therefore, to maximise P(all correctly classified), subject to the constraints that each phoneme (via a particular decoding method) is in one and only one cluster. In this embodiment, a Monte Carlo algorithm is used in order to determine phoneme classifications which maximise this probability.

Figure 9a illustrates the steps involved in this Monte Carlo algorithm. Initially, in step s200, the number of

phoneme clusters that will be used is determined. As those skilled in the art will appreciate, if there are too few clusters then there will be insufficient discrimination and if there are too many clusters then the data may not be retrievable. In this embodiment, in order to provide classifications which are sufficiently discriminative, ten clusters are defined. Once the number of clusters has been determined, the system randomly assigns, in step s201, phonemes to these clusters and stores this as a current configuration. The processing then proceeds to step s203 where the system determines the probability that the phonemes are correctly classified in the clusters for the current configuration, i.e. the system calculates the probability given in equation (5) above.

The processing then proceeds to step s205 where the system randomly selects a phoneme and a target cluster to which the selected phoneme may be moved. Then, in step s207, the system calculates what the probability given in equation (5) would be if the selected phoneme is moved into the target cluster. Then in step s209, the system compares this new probability calculated in step s207 with the probability for the current configuration which was calculated in step s203. If the new probability is

higher than the probability for the current configuration, then the processing passes to step s211 where the system moves the selected phoneme to the target cluster to replace the current configuration. The processing then proceeds to step s213 where the system determines whether or not the probability calculated for the new configuration is better than the "best ever" probability. If it is, then the processing proceeds to step s215 where the system stores the current configuration as the best ever configuration that it has encountered. Otherwise step s215 is skipped and the processing proceeds to step s219 where the system determines whether or not convergence has been reached.

If at step s209, the system determines that the new probability is not higher than the probability for the current configuration, then the processing proceeds to step s217 where the system moves the selected phoneme to the target cluster to replace the current configuration with a probability dependent upon the difference between the new probability and the probability for the current configuration. For example, a difference probability can be defined as:

$$d = \frac{1}{2} e^{-\lambda(s_1 - s_2)} \qquad (6)$$

where s_1 is the probability determined for the current configuration and s_2 is the probability for the proposed new configuration and λ is a parameter which can be tuned to get best performance. Therefore, when the two probabilities are the same $d = \frac{1}{2}$ and when the probability for the new configuration is massively worse than the probability for the current configuration d will be approximately equal to zero. Then if a random number generator is used which randomly picks a number between zero and one, then the proposed configuration will replace the current configuration if $d > r$, otherwise the proposed configuration is discarded.

After step s217, the system proceeds to step s219 where it determines whether or not convergence has been reached. If it has, then the processing ends and the phoneme clusters are stored. If convergence has not been reached, then the processing proceeds to step s221 where a random value (RV) between zero and one is chosen and then compared with a threshold (Th) in step s223. If the random value, RV, is less than the threshold then the processing returns to step s205 above and the procedure is repeated. On the other hand, if the random value, RV is not less than the threshold, then the processing proceeds to step s225 where the best ever configuration

is copied into the current configuration and then the processing again returns to step s205. However, by setting the threshold Th to be close to one ensures that the best ever configuration is only copied into the current configuration very occasionally. As those skilled in the art will appreciate the processing of steps s221 to s225 is provided to try and ensure that the system does not remain stuck in a local minimum.

The inventors have found that performing this clustering algorithm for the English phoneme set with ten clusters and for a given user, when the decodings come from the automatic speech recognition unit 51, gives the following phoneme clusters:

c_1 = {aa ae ah aw eh ey uw}

c_2 = {ao l oy r w}

c_3 = {d dh t th}

c_4 = {ax ea er hh oh ow ua}

c_5 = {m sil}

c_6 = {b f p v}

c_7 = {s z}

c_8 = {ch g jh k sh zh}

c_9 = {n ng}

c_{10} = {ay ia ih iy uh y}

and when the decodings come from the phonetic transcription unit 75, gives the following phoneme clusters:

c_1 = {aa ae ah aw eh ey uh uw}

c_2 = {ao l oy r w}

c_3 = {d dh t th}

c_4 = {ax ea er hh oh ow ua}

c_5 = {m sil}

c_6 = {b f p v}

c_7 = {s z}

c_8 = {ch g jh k sh zh}

c_9 = {n ng}

c_{10} = {ay ia ih iy y}

As those skilled in the art will appreciate, the phoneme clusters for the text to phoneme transcription unit 75 are predominantly the same as those for the automatic speech recognition 51, with the exception of the "uh" phone which is in cluster c_1 while it is in cluster c_9 for the clusters of the automatic speech recognition unit 51. As those skilled in the art will appreciate, the clusters given above are given by way of example only. The precise clusters that are used will depend on the matching method used to compare the phonemes in the clusters.

ALTERNATIVE EMBODIMENTS

In the above embodiment, the document database 29, the annotation database 31 and the speech recognition unit 51 were all located within the user terminal 59. As those skilled in the art will appreciate, this is not essential. Figure 10 illustrates an embodiment in which the document database 29 and the search engine 53 are located in a remote server 60 and in which the user terminal 59 accesses the database 29 via the network interface unit 67 and 69 and a data network 68 (such as the Internet). In this embodiment, both the documents and the annotations are stored in the database 29. In this embodiment, the user terminal 59 can only receive voice queries from the microphone 7. These queries are converted into phoneme and word data by the automatic speech recognition unit 51. This data is then passed to the control unit 55 which controls the transmission of data over the data network 68 to the search engine 53 located within the remote server 60. The search engine 53 then uses the phoneme index to carry out a search in the database 29 in a similar manner to the way in which the search was performed in the above embodiment. The results of the search are then transmitted back from the search engine 53 to the control unit 55 via the data network 68. The control unit 55 then considers the

search results received back from the network and displays appropriate data on the display 57 for viewing by the user 39.

In addition to locating the database 29 and the search engine 53 in the remote server 60, it is also possible to locate the automatic speech recognition unit 51 in the remote server 60. Such an embodiment is shown in Figure 11. As shown, in this embodiment, the input voice query from the user is passed via input line 61 to a speech encoding unit 73 which is operable to encode the speech for efficient transfer through the data network 68. The encoded data is then passed to the control unit 55 which transmits the data over the network 68 to remote server 60, where it is processed by the automatic speech recognition unit 51. In this embodiment, the speech recognition unit is operable to only generate phoneme data which is then passed to the search engine for use in searching the database 29 using the phoneme index 35. The search results generated by the search engine are then passed, via the network interface 69 and the network 68, back to the user terminal 59. The search results received back from the remote server are then passed via the network interface unit 67 to the control unit 55 which analyses the results and generates and displays

appropriate data on the display 57 for viewing by the user 39.

In a similar manner, a user terminal 59 may be provided which only allows typed inputs from the user and which has the search engine and the database located in the remote server. In such an embodiment, the phonetic transcription unit 75 may be located in the remote server 60 as well.

In the above embodiment, the annotation database and the document database were separate. As those skilled in the art will appreciate, in some embodiments, the annotation database and the document database may form a single database. Additionally, the annotations in the annotation database may form the data to be retrieved.

In the above embodiment, the annotation database was searched using both phonemes and words. As those skilled in the art will appreciate, the phoneme index described above and its use in searching the annotation database may be used in a system which does not search using words as well.

In the above embodiment, a phoneme index was described.

As those skilled in the art will appreciate, the above technique can be used for features other than phonemes, such as phones, syllables or katakana (Japanese alphabet) or any other sub-word unit of speech. This indexing technique could also be used in other applications, such as the indexing of DNA sequences and the like.

In the above embodiment, a hash indexing technique has been described. As those skilled in the art will appreciate, other indexing techniques can be used in order to identify portions of the database for carrying out a detailed phoneme search using phoneme data generated from the input query.

In the above embodiment, the function defined in equation (1) above was used to define the mapping between the cluster trigram and the entry in the table. However, with this technique, if C[1] or C[2] or C[3] equals zero then this will yield zero. Instead, the function used could be:

$$(C[3] + K_c(C[2] + K_c C[1]))\bmod S \qquad (7)$$

or, for a general N-gram of length n:

$$\left(\sum_{i=1}^{n} K_c^{(n-i)}.C[i]\right) \bmod S \qquad (8)$$

In the above embodiment, the pointers stored in the index were of the form n[p,q,r], where p is the annotation for document p, q is the q^{th} block of nodes within that annotation and r is the r^{th} node within that block. As those skilled in the art will appreciate, other types of pointers could be used. For example, the count of the node since the start of the lattice or the time and rank (where more than one node have the same time) of the relevant nodes could be used.

In the above embodiment, when the query is being applied to the index, each of the query cluster trigrams were applied to the index and the appropriate entries found. These entries were then compared in order to identify portions of the phoneme lattice for further searching. Alternatively, the processing of the query cluster trigrams may be performed incrementally, i.e. identify all the entries for the first query trigram, then obtain those for the second query trigram and retain those which are close in time to those of the first trigram etc.

In the above embodiment, when identifying regions where successive cluster trigrams occur close in time, the

system compared the node numbers in the pointers which are retrieved. However, in some embodiments, it is better to compare the time offsets stored for the nodes. This is because in some applications, the lattice will have many branches and two nodes which are very close in time could have completely different node numbers. The aim of this part of the system is to identify cluster trigrams which are in chronological order within the annotation and which occur within a limited period of time of each other. In this way, if there is an error in a trigram, it will only result in two to three trigram misses. Consequently, the time leeway should be comparable to about four or five phonemes which is about 0.2 to 0.3 seconds.